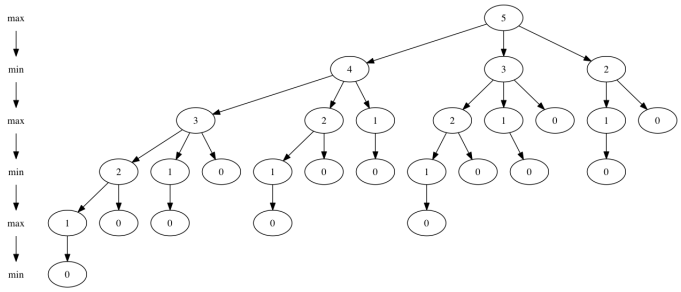
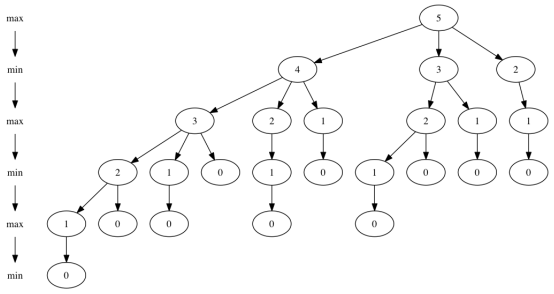
CSCE 625 Programming Assignment 2 Report

Peihong Guo (UIN 421003404)

October 8, 2014



(a)



(b)

Figure 1: Game tree of 5 matches: (a) using MinMax algorithm, 28 nodes; (b) using alpha-beta search algorithm, 25 nodes.

1 Introduction

In this project, two algorithms, MinMax search and Alpha-Beta search are implemented to play a simple match taking game. The game starts with a pile of n matches, where 2 players take turns to remove *one*, *two* or *three* matches from the pile. The player who removes the last matches loses the game.

2 Algorithm

The state of the game, s, is defined as a tuple (p, n) where p is the player (Max or Min) and n is the number of remaining matches. It is obvious that the terminal state of this game is the state with 0 match, i.e. $s.n = 0$. The utility for terminal state is defined as:

$$f(s) = \begin{cases} 0 \text{ if s.p=Min} \\ 1 \text{ if s.p=Max} \end{cases}$$

The utility is chosen such that the first player always maximize utility and the other player always minimize it.

2.1 Min-Max Algorithm

The Min-Max search algorithm recursively search for optimal action at a given state: if the player is Max, it chooses the action that maximizes the utility returned from evaluating all possible moves; otherwise it chooses the action that minimizes the returned utility. At terminal states, the utility value is obtained with the definition in previous section.

2.2 Alpha-Beta Search Algorithm

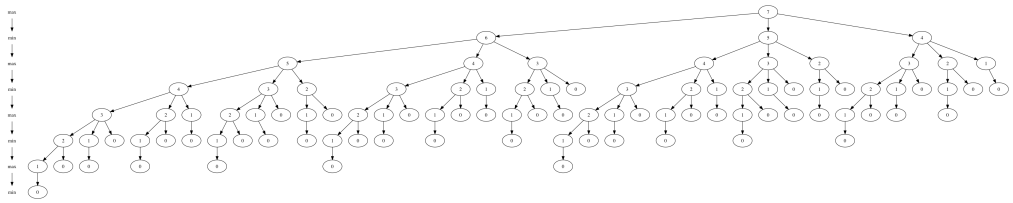
Alpha-beta search is a variant of Min-Max search that prunes subtrees that do not affect decision making for current state. To do that, two values, i.e. alpha and beta, are record in the min-max search process to denote the best value ever found for player Max and Min. Any action that produces worse result the the best found result will not be investigated further, therefore reducing the total number of actions for evaluation. The experiments run in this project demonstrated the effectiveness of alpha-beta pruning.

3 Analysis

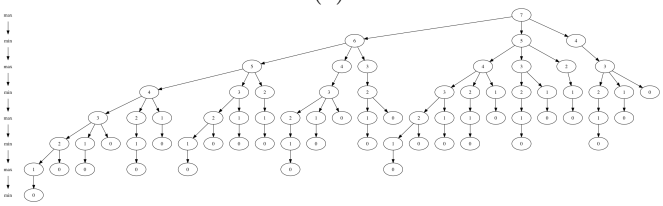
3.1 Comparison of Algorithms

Number of Matches	MinMax	Alpha-Beta Search	Relative Size
5	28	25	0.89285
7	96	71	0.73958
15	12640	3241	0.2564
21	489396	66341	0.12956
23	1655616	137302	0.08292

Table 1: Comparison of game tree size by MinMax algorithm and Alpha-Beta search. The size of a game tree is defined as the total number of nodes in the tree.



(a)



(b)

Figure 2: Game tree of 7 matches: (a) using MinMax algorithm, 96 nodes; (b) using alpha-beta search algorithm, 71 nodes.

A comparison of the performance of MinMax algorithm and Alpha-Beta algorithm is listed in Table. 1. It is clear that the alpha-beta pruning produce much smaller game trees. For a game with 21 matches, the tree by alpha-beta algorithm is only about 13% of the tree by min-max algorithm; the relative size of the alpha-beta game tree is even smaller than 10% of that by min-max algorithm for a game with 23 matches.

Figure. 1 and 2 shows the game trees generated by both algorithms for a game with 5 and 7 matches, respectively.

3.2 Optimal Strategy and Moore Machine

An interesting question about this game is whether opting to go first plays a role in the result of the game. It turns out the answer is yes, assuming both players are playing optimally. By running producing game trees for games with 5 to 21 matches, a pattern becomes clearly visible: Without doubt, for $n = 1$

n	Who Wins	n	Who Wins	n	Who Wins	n	Who Wins
5	second	9	second	13	second	17	second
6	first	10	first	14	first	18	first
7	first	11	first	15	first	19	first
8	first	12	first	16	first	20	first

and $n = 21$, the player goes second wins. That said, depending on the initial number of matches, going first or second will lead to opposite outcome in an ideal game.

In case of an optimal player against a random player, the optimal player can always win. The reason is the optimal player could choose to go first or second

based on the initial number of matches, and for each initial state, the outcome of the game can be completely determined by one player.

To see the reason why the assertion above is true, let's consider the optimal strategy of the game first.

From Table. 1 we may conjecture that the second player can always win if the initial state has $n = (4k + 1)$ matches, otherwise the first player can always win. This is in fact true.

Consider the initial state with only 1 match, the first player is guaranteed to lose, so the second player must win.

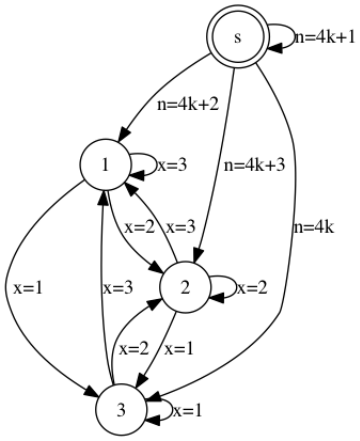
For initial state with 2, 3, 4 matches, the first player can always force the second player to remove the last match by removing 1, 2, or 3 matches. Therefore, the first player is guaranteed to win in these cases.

Consider the initial state with 5 match, the first player is again guaranteed to lose. This is because no matter how many matches the first player removes, the second player can always remove just enough matches to leave only 1 match in the pile. Suppose the first player removes m matches, the second player can to remove $4 - m$ matches. In total, they always remove 4 matches in the first two rounds, leaving the last match for the first player to remove.

This is also true for initial states with $4k + 1$ matches (5, 9, 13, 17, 21, ...), when the second player can always win by using the strategy describe above.

For initial state with $4k + l$ matches ($l = 0, 2, 3$), the first player is therefore guaranteed to win by removing $[(l - 1) \mod 4]$ matches in the first round, leaving $4(k - 1) + 1$ matches in the pile. This sets the second player in the case with $4k + 1$ matches initial state, in which the player go first would always lose. To win the game, the first player only needs to remove $4 - x$ matches in later rounds, following the strategy mentioned above, where x is the number of matches the second player removes.

To sum up, the optimal strategy for playing this game can be described with the Moore machine below:



The labels on the edge represents the opponent's move, and the node s represents the initial state. The number in each node represents the number of matches to

remove for current action, i.e. the output of the Moore machine. For example, if the current state is node 1, the next state is still node 1 if the opponent removes 3 matches; if the opponent removes 2 matches, the next state is node 2 (i.e. remove 2 matches); otherwise the next state is node 3. The looping edge of node s represents that in case of $n = 4k + 1$, let the opponents move first and make decision by treating the resulting pile as initial state.